D3 Workshop Part II: D3 Fundamentals Walkthrough

This is a step-by-step walkthrough of the second part of the workshop, introducing you to coding using D3.js. We will create a simple bar chart with one axis, with the colors of the bars controlled by the data values.

The goal of this workshop is not to reproduce this bar chart, but to understand how the steps taken to reproduce this bar chart can be applied to other visualizations as well. Please refer to the D3 Workshop Series Resources for some helpful tips and a list of other resources, guides, and tutorials that might come in handy on your data visualization journey.

Feel free to contact Payod Panda at ppanda@ncsu.edu to report any discrepancy or for further guidance.

1. Open the initial codebase that we'll start with, at go.ncsu.edu/d3_part2.

   The code template above has comments to describe each step. Go through the comments to get an idea of what the structure of the code will be like.

   I will refer to the number at the beginning of the comment blocks as the step. For instance, step 1 is defining the data variable, step 2 is defining the margin variable.

2. Let's first define the width and height of the chart area. We want the chart to be 640 px wide and 480 px high. Under step 3, add:

   ```
   var w = 640,
       h = 480;
   ```

3. Now we will create an SVG canvas in our HTML document. Under step 5:

   ```
   d3.select('body').append('svg');
   ```

4. We want this canvas' width attribute to match the w variable we defined above, and height to be h:

   ```
   d3.select('body').append('svg')
         .attr('width', w)
         .attr('height', h);
   ```

5. The bulk of our visualization will happen under step 6.
   Let's start by creating an enter() selection of our data and creating a data join between them and some rectangles. Please refer to Mike Bostock's excellent explanation on data joins and the enter(), update(), and exit() selections here.

   ```
   d3.select('svg').selectAll('rect')
         .data(data1).enter().append('rect');
   ```

6. The above code will create a new rectangle SVG element for every data point that hadn't been joined to an object yet. In this case, that means every data sample. Here's a breakdown:

a. `d3.select('svg')`
   Selects the SVG canvas element.

b. `.selectAll('rect')`
   Selects all the rectangle elements within the SVG canvas. At the moment, this is an empty selection that is a child of the SVG canvas because no rectangles exist.

c. `.data(data1)`
   Tells D3 that we want to use the data1 array variable as our data source.

d. `.enter()`
   Tells D3 that we want to access the enter() selection of this data source. Again, Mike Bostock's Thinking with Joins is excellent.

e. `.append('rect')`
   Appends a rectangle for every data point within the enter() selection above.

7. So the above code adds 11 rectangles (because we have 11 data points in the data1 array), but we haven't defined any of its attributes so we wouldn't see anything yet. Let's define the rectangle's required attributes. The cheatsheet for some common SVG elements can be found in the D3 Workshop Series Resources document.

```
d3.select('svg').selectAll('rect')
      .data(data1).enter().append('rect')
            .attr('x', 0)
            .attr('y', 0)
            .attr('width', 200)
            .attr('height', 50);
```

8. This will create 11 rectangles with width 200px, height 50px, located at (0,0). We only see one rectangle because all the 11 rectangles are at the same location and are the same size. We want to map the width of the

rectangle to the data points we have. Let's change the width attribute line to:

```
.attr('width', function(d){return d;})
```

Above we changed the width attribute to the return value of an anonymous function that takes a data point as an input parameter. Any time after d3's data() method is called, we have access to two variables; d and i. d denotes the data point currently being looked at, and i is its index in the dataset. We will use the index in some time.
The above code takes the data point as input, and returns the value of the data point. This controls the width of the rectangles.

9. At this point we still only see a single rectangle, because we haven't vertically staggered them yet. Let's control the y location of the rectangles by the index:

```
.attr('y', function(d, i){return i*50;})
```

10. You'll notice that we now have distinguishable rectangles, but they don't fit in the height. So we will declare a new variable that dynamically changes according to how many data points we have in the dataset (data1.length), before we add the rectangles. Under step 6:

```
var barHeight = h/data1.length;
```

11. We then change the height attribute to match this, and control the y position accordingly. So the attributes become:

```
.attr('x', 0)
.attr('y', function(d, i){return i*barHeight;})
.attr('width', function(d){return d;})
.attr('height', barHeight-2);
```

12. We want our bars to fit the chart's width. Let's define a scaling function so that the bars automatically adjust to the width of the chart, for instance if we add a new data point with a value of 700. Under step 4:

```
var xScale = d3.scaleLinear()
                    .domain([0, d3.max(data1)])
```

```
                .range([0, w]);
```

In the above lines, we define a function xScale, which is a linear scale. We define its domain to be from 0 to the maximum value in our data1 dataset, and we want it mapped to values ranging from 0 to the width of the canvas. So, a value of 0 in the dataset will be mapped to 0 px, and the maximum value in our dataset (which is 85 for our starting dataset) will be mapped to w (which is 640 in our case).

13. Now let's use the xScale function to control the width of our bars. Going back to the rectangle attributes,

    ```
    .attr('width', function(d){return xScale(d);})
    ```

    xScale(d) takes the data point value as a parameter and returns the scaled value according to our defined scale. The basic chart is now done! Let's make some stylistic changes to the chart.

14. To change the fill color of the rectangles, we can change the fill attribute:

    ```
    .attr('fill', 'orange')
    ```

15. Or we can be a little more advanced and use the data values to control the color. If the value is over 50, then color the bar blue, otherwise purple:

    ```
    .attr('fill', function(d, i){return d>50?
    '#63BAFC':'#9D6DE0'})
    ```

16. Let's now add an axis to our chart. For this, we will shrink the chart by a set margin, while keeping the canvas size the same. Look under step 2 to see the margins we have defined. Now go to step 3, and change the w and h variables like so:

    ```
    var w = 640 - margin.left - margin.right,
        h = 480 - margin.top - margin.bottom;
    ```

17. This will shrink the chart size to the correct dimensions, but also shrink the canvas size. We should add the margin values to the SVG canvas' width and height attributes. Under step 5 where you defined the SVG element, change the attributes:

```
d3.select('body').append('svg')
.attr('width', w + margin.left + margin.right)
.attr('height', h + margin.top + margin.bottom);
```

This will make the chart size smaller while retaining the same canvas size. However the changes wouldn't be visible yet because we haven't modified the location of the bars yet. What we will do is group all the elements under the chart under an SVG grouping element—the <g> element. Then we'll move this <g> element to where it needs to be, and our chart scaling will be done.

18. Under step 5, change the SVG canvas initiation to:

```
d3.select('body').append('svg')
.attr('width', w + margin.left + margin.right)
.attr('height', h + margin.top + margin.bottom)
  .append('g')
     .attr('transform',
'translate('+margin.left+','+margin.top+')')
     .attr('id', 'chart');
```

Here we appended a <g> element to the <svg> canvas element, and translated it to (margin.left, margin.top). We also added the id "chart" to this <g> element for easier identification later.

19. Now we need to change the hierarchy of the rectangle bars so they are created under this <g> element instead of the <svg> element. Under step 6, change the bar initiation code to:

```
d3.select('#chart').selectAll('rect')
        .data(data1).enter().append('rect')
        .attr('x', 0)
        .attr('y', function(d, i){return i*barHeight;})
        .attr('width', function(d){return xScale(d);})
        .attr('height', barHeight-2)
        .attr('fill', function(d, i){return d>threshold?
'#63BAFC':'#9D6DE0'})
```

20. Now let's create an axis. D3 has convenient built-in methods to create axes. We already have a scale defined for the x-axis, when we defined xScale. Let's use that to define the axis. At the end of the code, add:

```
var xAxis = d3.axisBottom().scale(xScale);
```

This line creates a new axis called xAxis, which will have the marking on the bottom side (axisBottom()). The markings will be governed by xScale.

21. Now let's add this axis to our canvas. After you define xAxis, add:

```
d3.select('svg').append('g')
        .attr('transform', 'translate(' + margin.left
+ ',' + (margin.top + h) +')')
        .call(xAxis);
```

This line appends a new <g> element to the <svg> canvas, translates it to where we would want the x-axis to be, and then calls the xAxis we defined earlier.

That's it! This should give you a basic horizontal bar chart with one axis denoting the value of the chart. You should also have differing colors of the bars dependent on the value of the data point being visualized. For review, here is the final code with some additions (link). We shall leave the added parts as homework—please use the reference document (link) and the resources within. The goal of this workshop series is not to reproduce this bar chart, but to be able to work with data and reproduce whatever visualization one wants. We hope this was useful.

Please feel free to contact the NCSU Libraries visualization services for more assistance, or Payod Panda at ppanda@ncsu.edu for questions specific to this workshop.